De "faire" à "faire faire": au coeur de la programmation: quelques réflexions didactiques

Duchâteau, Charles

Publication date:
1992

De "FAIRE" à "FAIRE FAIRE" : au coeur de la programmation : quelques réflexions didactiques.

Version française de

From "DOING IT ..." to "HAVING IT DONE BY ...":
The Heart of Programming.
Some Didactical Thoughts.

Conférence présentée au NATO Advanced Research Workshop

"Cognitive Models and Intelligent Environments for Learning Programming"

S;Margherita, Italie, 17-21 mars 1992

Charles Duchâteau

CeFIS, Facultés N-D. de la Paix, rue de Bruxelles, 61, B-5000 Namur, Belgium

Tél : +32 81 725060 Fax : +32 81 725064 E-mail : cduchateau@bnandp51

De "FAIRE" à "FAIRE FAIRE" : au coeur de la programmation : quelques réflexions didactiques.

Charles Duchâteau

CeFIS, Facultés N-D. de la Paix, rue de Bruxelles, 61, B-5000 Namur, Belgium

Résumé : Le concept même de *problème* est extrêmement spécifique en programmation. Le problème est, non dans la *tâche* qui sert de point de départ, mais dans la nécessité de *faire faire* cette tâche *en différé* par un dispositif *exécutant*. L'approche didactique proposée ici trace une progression possible de l'apprentissage centrée sur ce "faire faire" et attentive à proposer un portrait sémantiquement riche des contraintes apportées par l'exécutant-ordinateur dans le monde des tâches à programmer.

Mots Clés : résolution de problèmes, didactique de la programmation, langages impératifs, représentations, métaphores.

1. Introduction

"Not many educational objectives can be accomplished if would-be learners are frustrated by unnecessary difficulties with the programming language and environment. It seems to the authors that far too much attention has been given on the debate on transfer of competence, and far too little to attacking the problems of ease of learning and ease of use."

Mendelsohn dans [11].

Les réflexions et propositions qui vont suivre ne sont ni celles d'un spécialiste de la psychologie cognitive, ni celles d'un expert en informatique. Elles sont à la fois l'effet d'une découverte et d'un apprentissage "sur le tas" de la programmation [1] et le résultat de recherches didactiques entreprises durant ces dix dernières années à l'occasion de formations d'adultes à l'analyse et à la programmation.

---

[1] Le seul enseignement dont j'ai [2] bénéficié, il y a 20 ans, se réduit à trois journées de matraquage "FORTRAN". J'ai cru (j'ai peine à l'avouer) pendant plus de 10 ans qu'un compilateur était un des meubles mystérieux dont on devinait l'existence au sein du "centre de calcul" et la longueur totale des listings de mes programmes erronés "écrits" (c'est à dessein que je n'emploie pas le mot "conçus") pendant ces 10 mêmes années doit dépasser la distance terre-lune.

[2] Même si tout son propos repose en fin de compte sur son expérience, ses réflexions et son vécu, je sais qu'il est de bon ton que l'auteur reste en quelque sorte absent, comme sujet, du discours scientifique qu'il est pourtant en train de tenir. Je déplore cette habitude et tenterai de me réfugier en tant que sujet de cette contribution dans quelques unes des notes de bas de page ...

Le public auquel sont destinées ces formations a été et reste essentiellement constitué d'enseignants, non informaticiens mais chargés pourtant du cours d'informatique au niveau de l'enseignement secondaire. L'objectif est donc non de former de futurs professionnels de la programmation, mais de fournir à ces enseignants, en peu de temps et sous forme assimilable et adaptable, des concepts et des méthodes de la programmation mais également de porter une attention constante à la méthodologie adoptée et aux démarches de découverte ou de construction de ces concepts[3] . En quelque sorte, "l'emballage" est aussi important que le "contenu".

Après de nombreuses années où l'apprentissage de la programmation s'est confondu avec celui d'un langage de programmation, on a, par un mouvement de balancier dont l'enseignement est coutumier, tenté de minimiser l'importance de ce dernier : l'essentiel n'était plus l'expression dans un langage mais l'*analyse* préalable *du problème*, la *démarche "top down"* tenant lieu de main-courante dans cette descente qui devait conduire du *problème* à résoudre au texte du *programme*. Mais cette marginalisation du langage, c'est le discours des experts qui ont à ce point maîtrisé ces mêmes langages (de programmation) que les syntaxes ésotériques de ceux-ci leur sont devenues naturelles et qu'ils ont pu longuement (souvent par tâtonnement) se forger, au delà des particularités de ces langages, les représentations mentales des contraintes du dispositif informatique dont ils sont l'expression et le reflet[4] . On le sait, et cela sera rappelé ci-dessous, programmer c'est exprimer un "faire faire" dans lequel les contraintes liées aux capacités de "l'exécutant" sont primordiales et omniprésentes. Peut-on décrire ces contraintes qui fondent le concept même de "problème de programmation" sans passer par la description d'un langage dans lequel on peut s'adresser à cet exécutant ? Qu'est ce qu'une démarche *descendante* quand on n'a pas dit clairement vers quoi l'on peut *descendre* et quand on peut considérer que le "*rez-de-chaussée*" est atteint ?

Cependant, il faut le reconnaître, les langages de programmation "évolués" ont été créés pour les programmeurs et non pour l'apprentissage de la programmation[5] . On a sans doute raison de ne pas focaliser l'attention sur les détails syntaxiques de ces langages, même si, malgré le discours officiel, les initiations à la programmation continuent *forcément* à faire la part belle au langage. L'algorithme ou l'analyse ne sont le plus souvent que l'expression en langue naturelle du texte d'un programme préalablement rédigé[6] . Est-on condamné à cette

---

3 Je reprendrais volontiers à mon propre compte la constatation de Jacques ARSAC dans [1] : "*J'ai pratiqué la programmation... J'ai été contraint de réfléchir à ce que je faisais, pour pouvoir en rendre compte à mes étudiants. J'ai été encore plus sévèrement obligé d'y réfléchir face à ce public d'une extrême exigence intellectuelle et qui ne vous fait aucune concession : les professeurs de lycée.*"

4 "*Experienced programmers forget just how many of the implicit conventions about programming they have absorbed in their exposure to a variety of programming languages.*" [3]

5 "*Bonar and Liffick say in some detail why they think current programming languages are unsuitable for novices. They point out that their "emphasis in economy of expression ... is misplaced in designing languages for novice programmers.*."" [11]

6 "*L'observation d'élèves débutant en informatique dans un cadre scolaire semble indiquer que les méthodes d'analyse enseignées ne sont pas vues par les élèves comme un outil pour résoudre leurs problèmes de programmation, mais comme un contrat (avec l'enseignant) qu'il faut respecter. L'expression de la phase de travail qui correspond à l'analyse apparaît assez souvent comme une paraphrase du texte du programme écrit, qui peut précéder, accompagner, voire succéder à l'écriture directe dans un langage de programmation.*" [13], p. 317.

hypocrisie qui tout en minimisant pour les apprenants l'importance du langage et en magnifiant les vertus de l'analyse les laisse démunis face à ces injonctions.

Cette contribution tentera de montrer que s'il est illusoire de faire l'économie d'un certain "langage", on peut cependant garder à l'analyse sa primauté et sa nécessité : il faut pour cela créer entre l'univers de la tâche et celui d'un langage de programmation particulier, des strates supplémentaires où la recherche de métaphores pertinentes permettront à l'enseignant ayant maîtrisé (!) la programmation de faire partager aux apprenants les images mentales qu'il aura peu à peu cristallisées à travers sa pratique et sa maîtrise des langages. "...the solution is to use an *intermediate representation*, one which minimizes initial difficulty but which lead into real programming." [11]

2. Tâche et problème

L'une des raisons essentielles mise en avant pour promouvoir un apprentissage de la programmation dans le cadre d'une formation générale comme celle dispensée par exemple dans l'enseignement secondaire est son aptitude à développer chez ceux qui l'apprennent et la pratiquent la capacité de *résolution de problème*s[7] . Et pourtant si l'on s'avise d'ouvrir l'un ou l'autre des ouvrages consacrés aux débuts d'une initiation à la programmation, on trouve accolés aux mots "*analyse*" et "*problème*" l'énoncé de tâches anodines comme

- "donner la date de demain sachant quel jour nous sommes aujourd'hui";
- "écrire un mot à l'envers";
- "dire si un nom est présent dans une liste";
- "lancer un dé jusqu'à ce qu'on obtienne trois 6 de suite et dire alors combien de lancers ont été nécessaires";
- "trier par ordre alphabétique une série de noms"...

La multiplication des exemples renforce le sentiment qu'on se trouve là face à une série de tâches ineptes et fastidieuses mais qui ne méritent certes pas de se voir qualifier de problèmes.

Et l'injonction de l'enseignant demandant "d'analyser le problème" qui consiste à "déterminer le plus grand de trois nombres" ou (plus difficile !) de "vérifier si un texte est un palindrome" n'amène en général chez les apprenants, incapables de deviner en quoi ces travaux stupides peuvent donner lieu à une "analyse", qu'un malaise embarrassé. Même la question "comment fais-tu ?" est le plus souvent inutile ou insensée, tant l'exécution des besognes évoquées est immédiate et inconsciente.

Nous savons tous que, puisqu'il est ici question de programmation, chacune de ces *tâches* va cependant donner naissance à un véritable *problème*. Il va s'agir en effet non d'effectuer ces traitements mais de les *faire faire* par un dispositif *exécutant*. C'est dans ce fossé entre un "*savoir faire*" généralement inconscient et le "*savoir faire faire*" qui va obliger à intégrer les contraintes de l'univers de l'exécutant dans le monde de la tâche que se nichent l'algorithmique et la programmation.

Programmer dès lors c'est

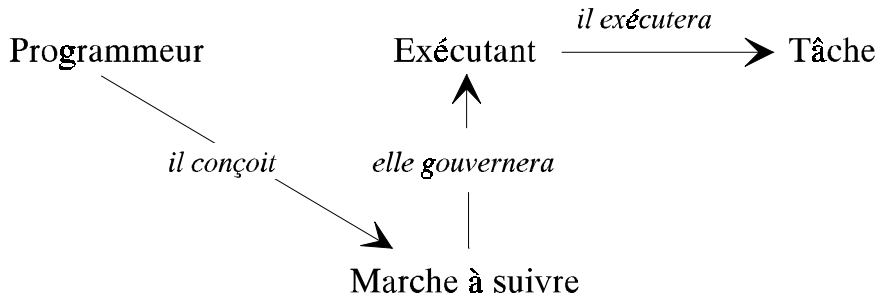
- face d'une part à une *tâche* (qu'il s'agira de bien définir et de préciser),

---

7 "*There has been quite some discussion recently about the urgency of incorporating computer literacy in our school curricula. An argument frequently adduced in support of such a development is the notion that experiences of computer programming turn people into better problem solvers.*" [14]

- face d'autre part à un *exécutant* aux capacités limitées mais *connues*,
- de rédiger la *marche à suivre* qui permettra de *faire faire* cette tâche par cet exécutant.

La problématique de la programmation (au sens large) tient donc dans le schéma suivant :



C'est ce même schéma qui est au centre des diverses activités de programmation. On comprend par exemple pourquoi le "micro-monde Logo" place l'apprenant au coeur même de cette démarche de programmation. Le portrait de l'exécutant est ici particulièrement significatif et transparent : il s'agit d'une tortue capable d'obéir à quelques ordres de déplacement en laissant une trace colorée des trajets commandés. Le problème n'est donc pas de dessiner un cercle, une maison ou un bateau (ce qui n'aurait strictement aucun intérêt) mais de faire tracer ces figures en fournissant à l'exécutant-tortue les marches à suivre adéquates. L'apprenant ne dessine rien : il *fait* dessiner (en différé), donc il programme.

Notons au passage qu'ici, de manière immédiate et naturelle, une strate métaphorique (les caractéristiques de la tortue) vient prendre place entre le langage (Logo) et la tâche (dessiner), strate qui permet de transporter, non les contraintes du dispositif informatique proprement dit, mais plutôt les exigences imposées par l'univers de l'exécutant au sein du monde de la tâche, créant ainsi le problème spécifique de programmation (faire dessiner).

C'est le même schéma qui est également au centre des *problèmes* de la commande numérique des machines outils, de la conduite des automates programmables, de la robotique,...

Deux remarques importantes à ce niveau :

a. Le "faire faire" exigé par l'activité de programmation n'est pas un "faire faire" en direct où, en présence de l'exécutant, on fournirait les instructions nécessaires en réagissant "sur le tas" aux conséquences des actions commandées. Il faut à l'avance avoir complètement déplié la tâche à faire effectuer en une marche à suivre qui précise en détail, sans ambiguïté et de manière exhaustive les instructions destinées à l'exécutant et les indications permettant d'organiser la suite de ses actions.[8]
b. Pour qu'un problème soit bien posé en programmation il faut non seulement que la tâche soit correctement précisée, mais encore que *l'exécutant qui sera chargé de son exécution soit complètement décrit* : sinon il est tout bonnement impossible d'écrire la marche à suivre qui doit forcément s'appuyer sur la connaissance des caractéristiques de celui-ci.

---

[8] Cette différence entre la commande en direct d'un exécutant "visible", qui obéit et réagit au fur et à mesure et la rédaction d'une marche à suivre qui oblige à anticiper et à imaginer le comportement futur provoqué chez l'exécutant, en l'absence de ce dernier, est centrale. Elle est à nouveau bien perceptible en Logo où l'apprenti-programmeur peut ou bien guider une tortue, présente à l'écran, bougeant et traçant à mesure que des ordres lui sont donnés, ou bien pénétrer dans cet autre univers (celui de la création du texte du programme à l'aide de l'éditeur), où la tortue a "physiquement" disparu et dans lequel les instructions écrites n'ont pas d'effet immédiat et visible. On retrouve cette même problématique dans le fossé entre les commandes en direct et l'écriture de "macros" permise par certains logiciels.

Nous verrons ci-dessous ce que doit comporter cette description d'un exécutant pour que la conception de toute marche à suivre qui lui est destinée soit possible. Il s'agit ici d'un des aspects les plus pervers des débuts de l'apprentissage de la programmation où l'on enjoint souvent à l'apprenant de "procéder par affinements successifs" sans lui avoir clairement précisé quand le décorticage peut être considéré comme achevé. Un énoncé -et il y en a des milliers de cette sorte- qui se contente de décrire la tâche (à faire faire) sans préciser les "capacités" de l'exécutant concerné *n'est pas* l' énoncé d'un problème de programmation.

Si l'on admet que la programmation (de type impératif) est bien ce "faire faire en différé" décrit ci-dessus, on peut immédiatement tracer un itinéraire possible pour son apprentissage. Il s'agira d'abord de découvrir les composantes des marches à suivre et les règles présidant à leur écriture et de s'exercer à leur conception à l'occasion de la "programmation" de tâches pour des exécutants particulièrement rudimentaires (et qui n'ont aucun rapport avec ce que sera l'exécutant-ordinateur). L'objectif essentiel de cette étape est de faire découvrir les constituants essentiels des marches à suivre (qui resteront inchangés quand il s'agira de programmer l'ordinateur) et plus particulièrement de faire manipuler les structures de contrôle qui en forment l'armature.

Dans un second temps, on s'efforcera d'exposer les capacités de base de l'exécutant-ordinateur, selon le schéma commun à la description des exécutants présentés précédemment. Il s'agira bien entendu d'un portrait métaphorique, non de l'ordinateur-machine mais des grands traits de l'ordinateur perçu comme exécutant. En particulier, le langage à travers lequel on s'adressera à lui sera syntaxiquement flou et sémantiquement riche et porteur d'images.

Dans un troisième temps, on proposera quelques tâches simples qui permettront d'acquérir les premiers tours de main du "métier de programmeur" et de bâtir peu à peu sur le socle de l'image initiale de l'exécutant-ordinateur des représentations mentales correspondant à des type de tâches qu'il devient possible (grâce à la maîtrise de ces tours de main) de lui commander. Autrement dit, au portrait originel se superposent peu à peu des strates correspondant non plus aux "compétences" primitives de l'exécutant mais aux actions plus complexes dont l'analyse ne pose plus *aucun problème* et fait place à une expression presque mécanique. Comme l'ont bien montré plusieurs auteurs [1,3,8,10,...] ce sont ces représentations de plus en plus larges non de ce dont l'exécutant est capable mais de ce qu'on peut arriver à lui faire faire *sans problème* qui peu à peu forgent les compétences du programmeur expert.

3. Portraits des exécutants et écritures des marches à suivre

Il est classique de distinguer dans l'expression des algorithmes d'une part les instructions correspondant aux actions élémentaires dont l'exécutant est capable et d'autre part les instructions d'organisation, les structures de contrôle qui vont permettre de planifier et d'ordonner la succession des actions commandées. Notons qu'un des constituants essentiels de certaines de ces structures organisatrices (l'alternative, les répétitions) est la notion de condition dont l'exécutant est capable de tester le moment venu la valeur de vérité.

Ainsi, ce qu'il est impératif se savoir du dispositif exécutant, c'est

- la liste des actions dont il est capable et les instructions à lui donner pour provoquer l'effectuation de ces actions primitives,
- le répertoire des structures de contrôle permises (séquence, alternative, répétition, branchement, appel de procédure, appel récursif, ...), et
- le relevé des conditions qu'il est capable de tester.

Il est indispensable aussi d'indiquer "l'environnement de travail" de cet exécutant : c'est ce qui donnera pleinement *sens* à ce qui précède et créera le lien entre la tâche à faire faire et les compétences de "celui" chargé de l'exécuter.

3.1. Un exemple de description d'exécutant : le robot "compteur de gouttes"

L'environnement de travail est le suivant :

- un alignement de verres déjà plus ou moins remplis,
- un compte-gouttes que pourra manipuler le robot,
- un récipient rempli d'une quantité plus que suffisante (inépuisable !) de liquide, auquel le robot pourra venir remplir le compte-gouttes.

L'état initial du compte-gouttes est inconnu; il y a au moins un verre; il peut se faire que tous les verres soient déjà remplis.

La marche à suivre à rédiger pourra comporter les instructions suivantes :

- *choisis le premier verre* : quel que soit ce qu'il faisait à ce moment, à l'instant où cette instruction est rencontrée, le robot vient (ou revient) "s'occuper" du premier verre de la série;
- *passe au verre suivant*[9] : le robot quitte le verre dont il s'occupait pour s'occuper du suivant;
- *presse une goutte dans le verre choisi*[9] : il presse une goutte et une seule dans le verre dont il s'occupe alors;
- *remplis le compte-gouttes* : le robot remplit le compte-gouttes dans le récipient, sans pour autant, le cas échéant, perdre de vue le verre dont il s'occupe.

Les conditions suivantes pourront figurer dans les structures de contrôle adéquates :

- *tous les verres sont remplis*;
- *le verre choisi est rempli*;
- *le compte-gouttes est vide*.

De plus, les contraires de ces conditions pourront aussi être évaluées de même que les assemblages formés de plusieurs de ces conditions liées par les mots ET et OU.

Il reste à préciser les structures de contrôle permises. A ce stade, il est sans doute préférable pour découvrir un style précis de programmation (procédurale) qu'elles ne varient pas d'un exécutant à l'autre : ce peuvent être celles de la programmation "structurée" (séquence, alternative, répétition, appel de procédure), celles de la programmation "à GO TO" (séquence, branchement, branchement conditionnel), ...

La *tâche* consisterait à remplir la série de verres, le *problème* c'est de concevoir et de rédiger la marche à suivre qui indiquera au robot, sur base des capacités précises décrites (et elles seules), comment il peut remplir cette série de verres.

3.2. Quelques considérations sur cette approche

Plusieurs remarques s'imposent ici :

- Les seules "expérimentations" menées l'ont été avec un choix de structures de contrôle qui est celui de la programmation "structurée".[10]

---

9 Je trouve préférable de ne pas insister a priori sur les circonstances qui rendraient "impossible" les actions ainsi commandées. Ainsi, l'injonction "passe au verre suivant" posera problème lorsque le robot s'occupe du dernier verre de la série; "presse une goutte dans le verre choisi" est impossible si le compte-gouttes est vide ou si aucun verre n'a encore été choisi... Trop insister dans l'énoncé sur ces restrictions, c'est déjà fournir aux apprenants des éléments de solution.

- Il n'y a pas ici de fossé entre l'univers de la tâche et celui de l'exécutant; ce dernier possède visiblement les compétences ad-hoc pour venir à bout de cette tâche. Le seul problème c'est d'organiser à bon escient les actions et les tests nécessaires. On isole en quelque sorte "à l'état pur" la difficulté d'utiliser correctement les structures de contrôle retenues.
- Le monde de l'exécutant ne recèle pas encore de contraintes syntaxiques : il est sémantiquement riche par rapport à la tâche à (faire) accomplir et syntaxiquement peu contraignant. Les seules contraintes "syntaxiques" portent sur l'utilisation correcte des structures de contrôle qui peuvent selon le choix de représentation retenu[11] s'exprimer par des "mots", des schémas, ...
- On trouve déjà ici lors du dépouillement des solutions (souvent erronées) imaginées par les apprenants, bien que les difficultés spécifiques liées aux compétences particulières de l'exécutant-ordinateur soient absentes de ces problèmes, un certain nombre de constatations parfois relevées par d'autres études :
  - l'utilisation de la structure répétitive "TANT QUE" est plus malaisée que celle de la structure "REPETER";
  - la structure alternative est quelquefois perçue comme commandant implicitement une répétition et confondue en quelque sorte avec le TANT QUE;
  - la structure d'appel de procédure qui a été présentée à ce stade comme "instruction d'action complexe assortie d'une référence à une (marche à suivre) annexe explicative" est employée sans problème par les débutants; l'usage naturel de cette structure ne comporte évidemment aucune des difficultés "techniques" liées dans la suite aux concepts de portée des variables, de paramètre,...
  - la prégnance d'une "exécution mentale" essentiellement séquentielle rend difficile l'emploi des structures de contrôle; les débutants partent plus volontiers de la succession des actions et tests telle qu'ils l'imaginent en se voyant oeuvrer "à la place de l'exécutant" que de la tâche dans sa globalité[12] ;...

Sur un autre plan, celui des modalités et difficultés liées à l'approche adoptée, quelques constatations se dégagent :

- le caractère arbitraire de ce qui est considéré comme primitif chez les exécutants-robots pose parfois problème (Pourquoi "remplis le compte-gouttes" suffit-il alors que "remplis le verre" n'est pas permis ou constitue en tout cas l'énoncé d'une action complexe (appel de procédure) qui nécessitera un décorticage supplémentaire);
- ce détour par la programmation de quelques exécutants-robots recule le contact avec l'ordinateur dans une perspective de programmation; les attentes relatives à un travail "sur les ordinateurs" doivent être simultanément rencontrées à travers des utilisations comme celle d'un éditeur de texte (dont la maîtrise sera, le moment venu, indispensable);

---

10 Rappelons que l'objectif premier n'est pas pour moi de mettre en évidence les comportements des apprentis-programmeurs mais de former "au mieux" les enseignants à la programmation. Ce relevé de faits importants apparus lors de l'apprentissage n'est qu'un sous-produit de la démarche de formation.

11 Ce choix est important et a fait l'objet de nombreuses études mais il est hors de mon propos actuel.

12 "*Another characteristic of beginner's strategies could be seen as generating the statements by mental execution of the program.*" [9]

- les débutants "adultes" comprennent rapidement en quoi cette approche prépare la programmation de l'exécutant-ordinateur qui suivra et sont dès lors prêts à accepter le caractère artificiel des situations proposées; comme, très vite, ils perçoivent la difficulté réelle liée au problème de "faire faire" des tâches pourtant peu significatives par les exécutants décrits, chacun des énoncés constitue l'occasion d'une vraie "résolution de problème", au sens pris par ces mots en programmation;
- il est assez simple de faire découvrir les erreurs de certaines solutions en pointant un contre-exemple et en "mimant" l'exécution insatisfaisante qui en résulte; cette "correction" est cependant un pis-aller puisque, contrairement à un micro-monde comme celui de Logo ou de la "vraie" programmation, on ne dispose pas vraiment ici des exécutants-robots adéquats. La mise au point d'un système multimédia qui organiserait des séquences vidéos en suivant les injonctions des marches à suivre produites pour produire des présentations visuelles de l'exécution de celles-ci n'est pas impensable; la vraie difficulté réside cependant dans le dépistage "intelligent" des conditions initiales (états des verres, du compte-gouttes,... dans le cas de l'exemple décrit) susceptibles de montrer les erreurs de programmation de telle ou telle production : le jeu n'en vaut sans doute pas la chandelle ...

4. Portrait de l'exécutant ordinateur

L'étape suivante proposée aux apprenants consiste en la découverte des capacités de l'exécutant-ordinateur et cela selon un schéma et en des termes proches de ceux utilisés pour les exécutants-robots qui ont précédé. Trop souvent, la découverte des traits essentiels et des contraintes du dispositif à programmer se fait à travers l'apprentissage d'un langage de programmation spécifique : les images mentales ne sont au mieux qu'un sous-produit de cette initiation au langage où les aspects syntaxiques viennent compliquer l'acquisition des aspects sémantiques relatifs au dispositif. Il est indispensable de proposer à la fois des métaphores pertinentes et, surtout, un "langage" de commande du dispositif exécutant porteur de signification et peu contraignant sur le plan de la syntaxe. C'est l'attention à des détails qui pourraient paraître insignifiants aux programmeurs avertis qui vont faciliter une prise de conscience des contraintes et capacités de l'exécutant. "... *whatever the programming language, beginners have to learn the operating rules of what is called the "device" underlying the programming language*" [10]; "*It seems that existing notations too often act to raise barrriers against programming* ..." [8]. Le but ici est de proposer une découverte des règles opératoires elles-mêmes à travers un choix d'images et de mots adéquats.

4.1. Exécutant-ordinateur et ordinateur-machine

On a souvent mis en évidence la nécessité de décrire le "dispositif informatique" à l'origine des contraintes présentes en programmation pour faciliter l'apprentissage de celle-ci. Ce n'est évidemment pas l'ordinateur-machine (avec sa mémoire, son processeur, son système d'exploitation, ...) qu'il est pertinent de présenter ici, mais plutôt l'ordinateur équipé du compilateur (ou de l'interpréteur) du langage impératif qui va le "transfigurer". Ce qu'il faut c'est en tracer un "portrait robot" en une métaphore qui reprenne ses traits essentiels. Ceux-ci sont, dans un premier temps en tout cas, peu dépendants du langage procédural retenu.

Quelques uns des éléments du décor (de l'environnement) dans lequel évoluera l'exécutant-ordinateur ont évidemment un rapport avec l'architecture de l'ordinateur-machine : clavier, écran, ... Mais il est, je pense préférable, de rompre ici avec une interprétation trop

"hardware" du portrait présenté : il suffit que les caractéristiques métaphoriques de l'exécutant-ordinateur soient pertinentes pour l'objectif poursuivi[13] : l'apprentissage de la programmation.

4.2. Les traits essentiels d'un portrait de l'exécutant-ordinateur

Il est évidemment hors de question ici d'entrer dans le détail de ce portrait : rien de très original d'ailleurs dans cette description sinon qu'elle va se conformer au canevas adopté : environnement, actions et les instructions correspondantes, conditions et structures de contrôle permises. En bref donc :

4.2.1. L'environnement

4.2.1.1. L'exécutant est un gestionnaire de tiroirs

Il dispose d'une série de "casiers" ou de "tiroirs" étiquetés. Un tel tiroir est caractérisé par :

- L'étiquette qui y est attachée : inamovible, choisie par le programmeur et non manipulable par l'exécutant. Cette étiquette constitue le nom du tiroir et servira dans le texte de la marche à suivre à désigner, soit le tiroir lui même, soit son contenu.
- Le contenu du tiroir : ce qui y transitera, c'est une information (nombre entier, nombre réel, chaîne de caractères)[14] .
- Le type du tiroir qui détermine le genre d'information susceptible d'y prendre place (entier, réel, chaîne).

Il dispose également d'une énorme "malle à informations", dont il pourra, le moment venu et sur les injonctions de la marche à suivre le gouvernant, tirer n'importe quelle information de l'un quelconque des trois types cités.

4.2.1.2. L'exécutant peut "communiquer" avec l'extérieur

Son "local de travail" dispose en effet d'une porte-clavier à travers laquelle il peut recevoir des informations et d'une fenêtre-écran où il pourra en afficher.

4.2.1.3. L'exécutant dispose d'une table de travail portant un certain nombre d'outils

Ces outils vont fournir, pour peu que, le cas échéant, l'exécutant y entre des informations du type adéquat, une information nouvelle.

Ainsi il y a un outil SOMME, un outil DIFFERENCE, un outil NOMBRE AU HASARD,...

Parmi ces outils, il en est qui possèdent un statut particulier, ce sont ceux qui permettront au rédacteur des marches à suivre d'énoncer les conditions acceptables. Ce sont les outils qui fournissent le résultat de comparaisons d'informations : >, >, =, ...

4.2.2. Les instructions d'actions élémentaires

Elles sont seulement au nombre de trois :

- l'instruction de remplissage d'un tiroir

  Place telle information *dans* tel tiroir

---

13 Les descriptions plus "réelles" ou plus "vraies" de l'ordinateur en terme de mémoire, de bus d'adresses, de compteur ordinal, ... ne sont pas plus "correctes" : ce sont seulement des images d'un autre niveau, pertinentes si l'on poursuit d'autres objectifs. En définitive, un ordinateur, ce sont des électrons en mouvement ... et nous sommes tous bien conscients que surtout à ce niveau il s'agit bien d'un "modèle" !

14 Le concept d'information de type booléen est prématuré à ce stade. Le seul contact avec le type booléen se fait à travers les conditions, qui seront bien entendu dans la suite, même si cela pose toujours des problèmes, ramenées à leur statut d'expressions booléennes.

- l'instruction de réception d'information de l'extérieur
  Va à la porte, attends l'information qu'on va t'y fournir et place la dans tel tiroir
- l'instruction d'affichage
  Affiche à la fenêtre telle(s) information(s)

Il reste à préciser ce que recouvrent dans deux de ces instructions les termes "*telle information*".

Une information pourra être écrite comme :

- une constante, comme dans
  Place 1 *dans* (le tiroir) Total, ou
  Affiche 'Bonjour'
- le nom d'un casier (c'est évidemment le contenu qui est alors désigné)
  Place (le contenu de) Somme *dans* (le tiroir) Total
  Affiche (le contenu de) Message
- le résultat fourni par un outil (ce qu'on appellera une expression)
  Place (le contenu de) Somme + 1 *dans* (le tiroir) Somme
  Affiche (le contenu de) Somme - (le contenu de) Total

4.2.3. Les conditions

Ce sont les comparaisons d'informations permises par les outils cités, comme
Somme < Total
Nom = 'Dupont', ...

Comme pour tous les exécutants précédents, des conditions plus élaborées pourront être construites grâce aux mots ET et OU.

4.3. Quelques considérations sur cette approche

Le squelette auquel j'ai ci-dessus réduit la présentation a gommé bien des détails (Cf. [5] à ce propos). Il me faut insister sur l'importance probable de quelques-uns de ceux-ci. Il resterait à prouver que ces "intuitions d'enseignant" sont (ou non) corroborées par les performances des apprenants.

a. Il faut donner à voir les notions présentées sous forme métaphorique. J'ai à cet effet réalisé et expérimenté une série de vidéos, porteuses d'images donnant corps aux métaphores employées. Ainsi, lorsqu'on *voit* que la "consultation" d'un casier ne le vide pas (c'est seulement une copie du contenu dont l'exécutant se saisit) mais que par contre le remplissage d'un casier en chasse l'information contenue jusque là, bien des difficultés de manipulation relatives au concept de "variable" sont aplanies.
b. C'est toujours avec mauvaise conscience qu'en tant qu'informaticien nous nous éloignons des choix de vocabulaire et de notations consacrés par l'usage. Pourtant, il est peut-être préférable d'attendre que des images mentales pertinentes soient installées avant d'introduire les termes habituels. Ainsi, je préfère parler de "casier" ou de "tiroir" plutôt (plus tôt) que de "variable", de "remplissage" plutôt que "d'affectation", "d'étagère" plutôt que de "tableau",...[15]

---

15 J'ai jusqu'il y a peu, pour des raisons de conformité avec l'usage et dans un souci de concision, remplacé l'instruction d'entrée ou de réception d'information ("Va à la porte ...") par le raccourci "Lis et place dans ...". Cela donne bonne conscience, puisque, classiquement, on parle d'instruction de "lecture" et que bien des langages la codent en "read ...". Malgré toutes les précautions et avertissements cette "lecture" perturbe inutilement les débutants qui y attachent je ne sais quelle image mentale non pertinente. Plutôt que de continuer pendant des années encore à étudier les difficultés d'apprentissage causées (en partie du moins) par des environnements langagiers (entre autres) inadéquats, ne pourrait on consacrer une partie de cette

Il faut cependant signaler qu'une fois les images véhiculées par les périphrases et le vocabulaire installées, une expression plus concise sera adoptées. Mais elle se présentera non comme une contrainte extérieure mais plutôt comme une série de raccourcis ou de mnémoniques destinés à faciliter et fluidifier l'écriture.

c. Un certain nombre de détails sont probablement importants et mériteraient d'être évalués : ainsi, faut-il présenter un exécutant disposant de casiers "fermés" (on n'en voit pas plus que lui le contenu) [10] ? Le fait de désigner l'emploi des "outils" par des termes dénotant le résultat (LA SOMME DE...) plutôt que l'action ou le processus (ADDITIONNER...) est-il essentiel ? [16]

d. Il est un trait important de l'exécutant-ordinateur sur lequel il faut insister, c'est son caractère "amnésique". Je ne peux lui parler qu'à l'impératif présent[17] sans jamais faire référence au passé : sa seule "mémoire", à tout instant de l'exécution, c'est le contenu *actuel* des tiroirs dont je l'aurai doté. Des termes comme "tantôt", des temps comme l'imparfait, sont à bannir.

e. Un reproche qu'on pourrait faire à cette peinture de l'exécutant-ordinateur, c'est son caractère très anthropomorphe. L'excès d'images et de métaphores est ici le remède même à ce reproche : je n'ai à ce jour rencontré aucun apprenant, même débutant, qui imagine un seul instant que c'est vraiment "comme cela que ça se passe dans l'ordinateur". Mais très vite tous comprennent qu'on peut "faire comme si...".

f. L'environnement pour l'apprentissage de la programmation "Images pour programmer" [4], outre les vidéos et leurs documents d'accompagnement et l'ouvrage écrit, comporte aussi un petit logiciel (START) qui permet de suivre, dans un environnement schématique correspondant à celui employé lors de la description de l'exécutant, le déroulement de l'exécution des premiers petits programmes conçus par les apprenants. Bien que le langage permis soit fort proche de celui présenté ci-dessus[18] , deux limites importantes sont à signaler :
   - même si le langage utilisé est porteur de sens, on n'échappe pas à la nécessité du respect strict d'une syntaxe rigide, alors qu'à ce niveau cela reste accessoire;
   - il ne permet pas de découvrir le comportement de l'exécutant en mode "direct" : on ne peut donner des instructions et juger immédiatement de leur effet; on ne peut tester que de petits programmes, par le détours de leur écriture à travers un éditeur de texte.

énergie à étudier et tester des descriptions et des "langages" qui s'affranchissent enfin de ces expressions impropres ou trompeuses, de vrais langage non "de programmation" mais "pour apprendre la programmation". "*Certainly a number of difficulties for beginners can be drastically reduced by designing ergonomic programming language syntax*" [10].

16 Il est regrettable et fort dommage que la plupart des travaux émanant du champ de la psychologie de la programmation restent inconnus des enseignants d'informatique et des didacticiens. Il serait important que les chercheurs en psychologie "vulgarisent" les résultats de leurs investigations et que des dialogues s'engagent. La didactique de l'informatique doit se nourrir des apports de l'épistémologie (naissante) de l'informatique et de la psychologie de la programmation. Je dois avouer que bien que j'enseigne l'informatique depuis plus de 10 ans, la plupart des résultats de psychologie de la programmation m'étaient jusqu'il y a très récemment complètement inconnus. Je ne pense pas que cela soit exceptionnel...

17 Le temps qui conviendrait en réalité pour ce "faire faire en différé" n'existe pas : il faudrait un "impératif futur" !

18 "*... the implementation of the programming language and the preparation of teaching materials must be developed as a whole, so that each refers to the machine in the same terms*." [3]

Il faudrait sans doute élaborer des environnements d'*exploration* des capacités de l'exécutant-ordinateur avant de proposer des environnements de *programmation* de ce même exécutant.

g. Le portrait initial de l'exécutant est somme toute assez simple. C'est vrai que même si des actions supplémentaires viendront peu à peu s'ajouter aux trois présentées ci-dessus, elles ne modifieront pas de manière essentielle le portrait tracé. Les capacités réelles de cet exécutant sont bien entendu fonction des outils de "calcul" dont il dispose sur sa table de travail. Il est en tout cas inutile dans un premier temps de gaver les débutants de longues listes de ces outils dont ils ne sauraient que faire. Ces possibilités seront découvertes plus tard et au moment où les tâches à programmer les exigeront. Les fonctions et opérations s'appliquant aux données numériques sont d'ailleurs aisément admises sinon prévues par les apprenants. Il en va tout autrement des outils de manipulation des chaînes de caractères qui devront être présentés avec beaucoup de soin car les opérations qu'ils permettent n'ont pas leur équivalent dans les traitements qu'un être humain peut appliquer à du texte.[19]

5. Les premiers tours de main de la programmation

Les concepts essentiels sont maintenant présentés et disponibles; le plus important reste à venir : faire manipuler ces concepts pour que chaque apprenant se forge progressivement des images mentales supplémentaires et cela en découvrant et en maîtrisant les tours de main du "métier de programmeur"[20] .

C'est à présent que, peu à peu, l'apprenant doit, sur le socle décrit, bâtir ses représentations de l'univers de l'exécutant-ordinateur et des contraintes et possibilités afférentes. Il faut qu'il passe de ce qui est primitif à ce qui est possible. Peu à peu, il doit engranger des types d'actions ou de tâches qui sans correspondre à des capacités "natives" de l'exécutant ne posent plus de réel *problème* d'explicitation, de programmation : les tours de main correspondants sont acquis.[21]

Le rôle de l'enseignant reste important dans l'acquisition de ces enrichissements successifs du catalogue des tâches (ou des parties de tâches) qui deviennent anodines à programmer.

a. D'abord, et cela n'est pas le plus simple, il doit dresser une liste des "tours de main" à acquérir. C'est d'autant plus malaisé que l'introspection est difficile : le propre de la maîtrise d'un tour de main, c'est justement que ce dernier est à ce point intégré au savoir faire et aux compétences de l'expert qu'il en devient inconscient. Il est bien difficile d'isoler ce qui un jour posa problème mais est devenu presque réflexe.

---

19 On retrouve là une caractéristique commune à l'exécutant-ordinateur et à la machine-ordinateur : les traitements d'information y sont toujours *formels*. C'est parfait pour les nombres dont les manipulations sont par nature formelles; c'est beaucoup plus dérangeant pour les informations de type "textuel" : l'être humain ne manipule pas des *chaînes de caractères*, il se sert de *mots*... Nous sommes là au coeur de l'informatique, quête sans fin pour enfermer toujours davantage le sens dans la forme.

20 "*I suggest that a too early training of top-down programming methods should be avoided, before the subject has correctly learned the constraints of the computer operation on the overall structure of the procedures, and knows a wide range of concrete plans to be transferred*." [9]

21 "*These tasks cannot be considered as problems for professional programmers who can activate well-known computers schemas ready to fulfil them."* [10]

b. Cette liste étant dressée, il faut concevoir la description des tâches à programmer qui vont activer ces tours de main. L'idéal est bien entendu que, dans un premier temps et dans toute la mesure du possible, la tâche proposée isole "à l'état pur" le tour de main à faire acquérir. Il ne suffira évidemment pas d'avoir exercé une fois cette habileté pour qu'elle prenne place dans le stock mental du programmeur : le propre de l'acquisition d'un tour de main c'est justement de nécessiter la répétition de son exercice dans des contextes divers.
A titre d'exemple, je citerais parmi beaucoup d'autres :
- le tour de main du "casier compteur" avec comme tâche permettant une première mise en oeuvre "accepter une série de nombres en arrêtant lorsque 0 est fourni et annoncer alors combien de nombres ont été ainsi reçus";
- le tour de main du "casier signal" (qui trouve son origine dans le caractère amnésique de l'exécutant), avec comme tâche incitatrice "accepter (lire) des mots en arrêtant à la réception de "STOP" à condition que le mot "ATTENTION" ait été reçu auparavant"; ...

6. En guise de conclusions

Les 10 années qui viennent de s'écouler ont vu la généralisation d'un enseignement de l'informatique et, plus spécifiquement, de l'algorithmique et de la programmation au sein du niveau secondaire. Quelles que soient les approches didactiques adoptées, on se heurte généralement au caractère peu significatif et peu motivant des premières tâches abordées. Ce fossé croissant entre les performances et l'ergonomie des outils logiciels récents (que découvrent les apprenants) et les réalisations possibles au début de l'apprentissage de la programmation[22] est l'un des facteurs qui fait aujourd'hui souvent rejeter l'enseignement de cette dernière.

Par ailleurs, les concepts de la programmation impérative sont de plus en plus présents au sein même des logiciels-outils à travers l'écriture de macros ou via des langages de commande spécifiques. Ils sont indispensables aussi dans les activités de micro-robotique qui se généralisent. Nous avons là de merveilleuses occasions de faire maîtriser les concepts généraux de l'algorithmique dans des environnements qui ne soient plus ceux des langages de programmation classiques. Quel portrait du dispositif exécutant sera-t-il pertinent de tracer lorsque l'interlocuteur sera l'ordinateur équipé d'un tableur ou d'un logiciel de traitement de texte ? Que deviendront demain les difficultés spécifiques à la maîtrise des structures de contrôle quand les dispositifs exécutants auront été ainsi dépeints ? [2].

Il reste bien des questions ouvertes ! Là aussi, les approches didactiques auront à se nourrir des apports d'une psychologie cognitive qui devra s'intéresser moins aux langages classiques qu'aux aspects algorithmiques des nouveaux outils logiciels.

7. Bibliographie

[1] ARSAC J.
Préceptes pour programmer
Dunod, Paris, 1991.

[2] DAGDILELIS V., BALACHEFF N., CAPPONI B.
L'apprentissage de l'itération dans deux environnements informatiques.

22 L'élève achève d'utiliser un traitement de texte avec icônes, couleurs et autres menus déroulants et on lui propose l'écriture d'un programme qui fait inverser la première et la dernière lettre d'une chaîne de caractères...

ASTER, n° 11, Paris, 1990, pp 45-66

[3] DU BOULAY B., O'SHEA T., MONK J.
The black box inside the glass box : presenting computing concepts to novices
in Man-machine studies (International journal of) n° 14, 1981
pp. 237-249.

[4] DUCHATEAU C.
Images pour programmer. Un environnement pour l'apprentissage de la programmation.
CeFIS, Facultés. N-D. de la Paix, Namur, 1989.

[5] DUCHATEAU C.
Images pour programmer. Apprendre les concepts de base.
De Boeck-Wesmael, Bruxelles, 1990.

[6] DUCHATEAU C.
Incursion au pays du faire faire.
CeFIS, Facultés N-D de la Paix, Namur, 1983.

[7] DUCHATEAU C.
Programmer ! Pour une découverte des méthodes de la programmation.
Wesmael-Charlier, Leuze-Longchamps, 1983.

[8] GREEN T.R.G.
Programming Languages as Information Structures
in HOC J-M., GREEN T., SAMURCAY R., GILMORE D.
Psychology of Programming
Academic press, 1990, pp. 117-138.

[9] HOC J.-M.
Analysis of Beginners'Porblem-Solving Strategies in Programming
in GREEN T.R.G., PAYNE S.J., VAN DER VEER G.C.
The Psychology of Computer Use.
Academic Press, London, 1983, pp. 143-158.

[10] HOC J.-M. AND NGUYEN-XUAN A.
Language Semantics, Mental Models and Analogy
in HOC J-M., GREEN T., SAMURCAY R., GILMORE D.
Psychology of Programming
Academic press, 1990, pp. 139-156.

[11] MENDELSOHN P., GREEN T.R.G., BRNA P.
Programming Languages in Education : The Search for an Easy Start
in HOC J-M., GREEN T., SAMURCAY R., GILMORE D.
Psychology of Programming
Academic press, 1990, pp. 175-204.

[12] ROGALSKI J.
Enseignement de méthodes de programmation dans l'initiation à l'informatique
in Actes du colloque francophone sur la didactique de l'informatique.
Université René Descartes, Paris, 1, 2, 3 septembre 1988.
Editions de l'EPI, Paris, 1989, pp. 61-74.

[13] ROGALSKY J., SAMURCAY R., HOC J-M.
L'apprentissage des méthodes de programmation comme méthodes de résolution de problème
in Le travail humain n° 51, 1988, pp. 309-320.

[14] VAN DER VEER G.C., VAN DE WOLDE G.J.E.
Individual Differences and Aspects of Control Flow Notations
in GREEN T.R.G., PAYNE S.J., VAN DER VEER G.C.
The Psychology of Computer Use.
Academic Press, London, 1983, pp. 107-120.